EECS 1012: LAB 5 – More Computational Thinking (Feb 03–09, 2020)

A. REMINDERS

1) You **must** attend your **assigned** lab session (we will be marking your submission in the lab).
2) You must arrive on time: anyone later than 15 minutes will not be admitted to the lab.
3) You must complete the pre-lab mini quiz posted on Moodle in the first 15 minutes of your lab time.
4) Each lab including the pre-lab mini quiz is about 2% of your overall grade.
5) TAs are in the lab to help you. They will also verify and mark your work at the end. Signal a TA for help if you stuck on any of the steps below. Yet, note that TAs would need to help other students too.

B. IMPORTANT PRE-LAB WORK YOU NEED TO DO BEFORE GOING TO THE LAB

1) Download this lab's files and read them carefully to the end.
2) See the sample solution to Lab04.
3) You should have a good understanding of
    - Iterations: *while*, *do-while* and *for* loop symbols as introduced in the lecture notes
    - JS operator for division and modulus operator
    - JS prompt box
4) Practice drawing flowchart symbols in draw.io or MS Word or PowerPoint. If you plan to draw your flowcharts on paper, please make sure you have pencils, erasers, and perhaps a ruler.

C. GOALS/OUTCOMES FOR LAB

- To practice more on computational thinking and implement the solutions in JavaScript.

D. TASKS

1) You first and major task in this lab is to draw seven flowcharts. This task must be done in teams of two (not fewer, not more). When you are done, you should show your flowcharts to the TAs before you go to next part. The TA may ask you to make minor modifications to your flowcharts to demonstrate your computational thinking skills in those contexts.
2) Then, you are provided with `ct.html`, document and supporting files such as `ct.css` and `ct.js`. Your task is to translate your seven flowcharts to JavaScript code.
3) You will generate at least five `html` and `js` files in this process. You should demo each HTML file to the TA. For that, please, have each `html` file open in a different tab so you can show the progression.
4) See the following few pages for details on how to modify your `html` and `js` files.

E. SUBMISSION

1) Manual verification by TA

You will need to have one of the TAs verify your lab before submission. The TA will look at your various files in their progression. The TA may ask you to make minor modifications to the lab to

demonstrate your knowledge of the materials. The TA will mark your name off a list; You are required to sign the list to show your attendance and that you have been verified.

2) Moodle submission

You will see an assignment submission link on Moodle. Create a **folder** named "**Lab5**" and copy all of your lab materials inside (`img_{09,10,11,12,13,14,15.jpg`; `ct_Ex{9,10,11,12,13,14,15}.html` and `ct_Ex{9,10,11,12,13,14,15}.js`). This folder should be compressed (or tar.gz) and the compressed file submitted.

F. COMPUTATIONAL THINKING

Part 1: This exercise must be done in teams of two (with the permission of the TA, only one team can be in team of three if the lab population is odd). If you have done it at home, you must discuss it with a peer from your lab before you show your final solution to your TA.

Using a computer program (or a pen and pencil), draw the following flowcharts and write your name on each. By end of this lab, you should take a screenshot (or a picture) from each flowchart and both you and your teammate should submit them to Moodle as **img_**{09,10,11,12,13,14,15}**.jpg** files, where **img_x** is the flowchart of exercise **x** below. Try to keep the size of each image below 500 KB, e.g., by reducing the resolution of your camera.

IMPORTANT: You are required to use the symbols introduced in the lecture which are inspired from this book ("Computer Science: a first course" by Forsythe, Keenan, Organick, Stenberg).

IMPORTANT: You are required to provide preconditions and postconditions for each solution you provide.

IMPORTANT: In Ex 9 to 12, you are not allowed to use strings. Instead, you should work with numbers and math operators, such as division, modulus, etc.

Ex 9) Devise a flowchart to receive a positive number and output each digit separately. For instance, if the input is 692, the program should output 2, 9, 6. Another example, if the number is 135429 the program should out put 9, 2, 4, 5, 3, and 1.

Ex 10) Devise a flowchart to receive a positive number and output how many of its digits are equal to 7. For instance, if the input is 772, the program should output 2, because there are two sevens there. Another example, if the input is 14368, the program should output 0.

Ex 11) Devise a flowchart to receive a positive number and output sum of its digits. For instance, if the input is 63932, the program should output 23, because 6+3+9+3+2 is 23. Another example, if the input is 23 the program should output 5.

Ex 12) Devise a flowchart to receive a positive number and output "yes" if it's equal to its reverse; otherwise, output "no". For instance, if the input is 63936, the program should output "yes", because if you read the digits from left to right or from right to left, it's the same number. But, if the input is 632, the program should output "no" because 632 is not the same as 236.

Ex 13) Devise an algorithm to receive a positive number, as *n*, and output *n!* (*n* Factorial).

Ex 14) Devise an algorithm to input an integer greater than 1, as *n*, and output the first *n* values of the Fibonacci sequence. In Fibonacci sequence, the first two values are 0 and 1 and other values are sum of the two values preceding it. For instance, if the input is 4, the program should print 0, 1, 1, 2,. As another example, if the input is 9, the program should output 0, 1, 1, 2, 3, 5, 8, 13, 21,.

By the way, after you are done with Ex 14, learn more about applications of Fibonacci number in real life here: http://www.ijesi.org/papers/Vol(6)9/Version-3/B0609030714.pdf . There might be a question on this in the pre-lab mini quiz.

Ex 15) Devise an algorithm to input a positive integer, *n*, – and by using XX characters – output the figure below that has *n* rows and each row *k* has *k* pairs of XX. For instance, if input is 5, the figure on the left (and if the input is 12, the figure on the right) should be generated by the program.

```
number: 5

XX
XX XX
XX XX XX
XX XX XX XX
XX XX XX XX XX
```

```
number: 12

XX
XX XX
XX XX XX
XX XX XX XX
XX XX XX XX XX
XX XX XX XX XX XX
XX XX XX XX XX XX XX
XX XX XX XX XX XX XX XX
XX XX XX XX XX XX XX XX XX
XX XX XX XX XX XX XX XX XX XX
XX XX XX XX XX XX XX XX XX XX XX
XX XX XX XX XX XX XX XX XX XX XX XX
```

Show all your flowcharts to your TA before going to Part 2. The TAs may ask any teammate some questions about the flowcharts, they may also ask you to modify your flowcharts slightly.

Part 2: you are given **ct.html**, **ct.css**, **ct.js** files. By reading these files carefully, you can enhance your learning before diving to details of exercises below. In this part, you translate your flowcharts of Part 1 to JavaScript code.

Exercise 9. Copy **ct.html** to a new file named **ct_Ex9.html**. Copy **ct.js** to a new file named **ct_Ex9.js**.

Launch **ct_Ex9.html** with your browser and click on the button, nothing happens. In this exercise, you translate your flowchart of Ex 9 of Part 1 to its equivalent JavaScript code. First let's fix some html code:

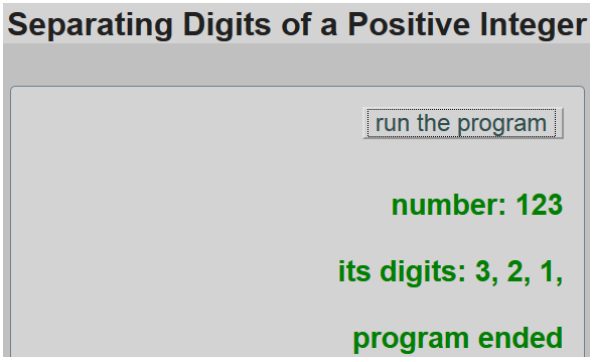
Make four changes to **ct_Ex9.html**, as follows:

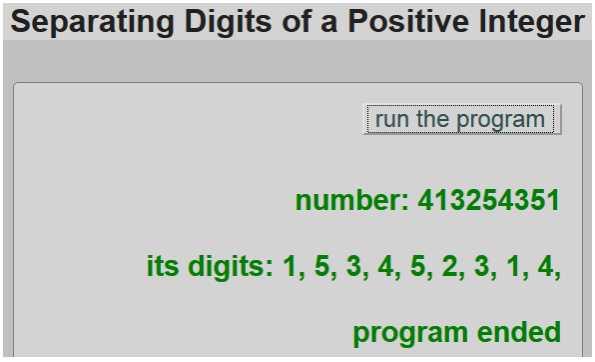
1) Connect it to **ct_Ex9.js** by fixing the script tag in the head element.
2) Correct the header `h1` of the document to show "Separating Digits of a Positive Integer"
3) Correct the name of the event function of the button to be "problem_09()"
4) Add your name to the list of authors of this page.

Also, follow these guidelines in your **ct_Ex9.js**:

- Make sure name of the function is proper.
- Part of the function has been provided for you; if you run your program at this point, it receives a number, but it does not separate the digits, and it stops. You should use your flowchart that you drew in Part 1 to complete the function.
- In your flowchart, you should have a loop, perhaps a `while` loop (or a `do-while`), translate it to `js` starting from line 20 of **ct_Ex9.js**.

Once you are done, run the program, you should see the following results if you enter 3498 or 100123873. If not, debug your code (*Shift+Ctrl+J* in Firefox).





Before going to next exercise, make sue your JS code is a good match to your flowchart for this problem. If not, you should fix this mismatch.

Exercise 10. Copy **ct_Ex9.html** and **ct_Ex9.js** to new files named **ct_Ex10.html** and **ct_Ex10.js**.

In this exercise, you translate your flowchart of Ex 10 of Part 1 to its equivalent JavaScript code.

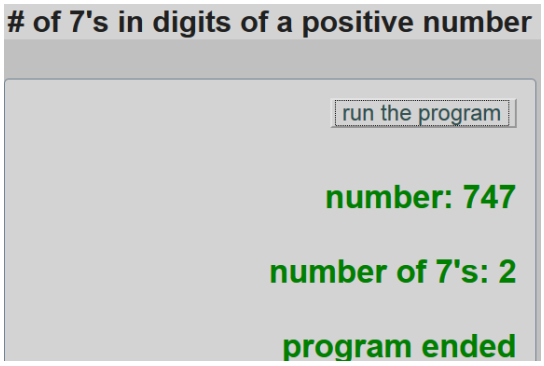
In the `js` file of previous exercise, you (should have) separated digits of a number by modulus and division operators. Now, in **ct_Ex10.js**, you need to make the following changes

- Name of the function should be `problem10()`
- In line 15, change the output message from "its digits" to "number of 7's"
- Before entering the loop (e.g. in line 19), declare a `counter` variable and assign 0 to it.
- Inside the loop, after you separated a digit (by modulus operator) check if it's equal to 7 or not, if yes, add to the `counter`.
- You perhaps had an output box inside the loop in your flowchart of Exercise 9; and, you probably do not have an output box inside the loop in you Exercise 10's flowchart. Instead, you should have an output box after the loop iterations are over to show how many 7's you've counted. If that's the case, translate that output box to a `js` statement like this:

```
outputObj.innerHTML=outputObj.innerHTML+counter;
```

Also, you need to make 3 changes in your **ct_EX10.html**: script, header h1, and name of the event function.

Once you are done, run the program, you should see the following results if you enter 789041 or 5323653345. If not, debug your code (*Shift+Ctrl+J* in Firefox).



Before going to next exercise, make sue your JS code is a good match to your flowchart for this problem. If not, you should fix this mismatch.

Exercise 11. Copy **ct_Ex10.html** and **ct_Ex10.js** to new files named **ct_Ex11.html** and **ct_Ex11.js**.

In this exercise, you translate your flowchart of Ex 11 of Part 1 to its equivalent JavaScript code.

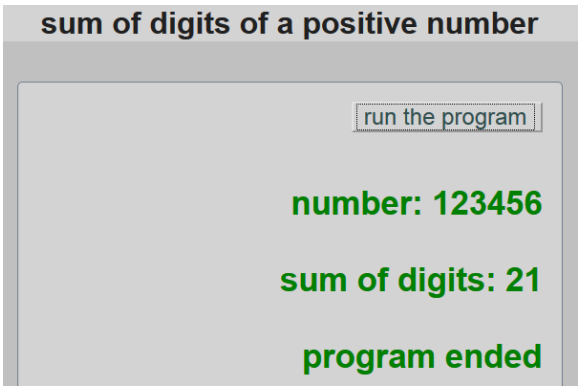
First, in **ct_Ex11.js**, you need to make the following changes

- Name of the function should be `problem11()`
- In line 15, change the output message from "number of 3's" to "sum of digits".
- In line 19 of **ct_Ex10.js**, you had declared a `counter` variable and assigned 0 to it. Just change name of that to `sum`, because here you do not count anything, you just add the digits together.
- In your flowchart of Ex 10, inside the loop, after you separated a digit (by modulus operator), you checked if it's equal to 3 or not. We no longer need that `if` statement here. Instead, add the separated digit to `sum`.
- You have an output box after the loop iterations are over; but, you are not going to output `counter` any more, you output `sum`. In that case, the statement should look like this:

```
outputObj.innerHTML=outputObj.innerHTML+sum;
```

Also, you need to make 3 changes in your **ct_EX11.html**, similar to those you made in **ct_Ex10.html**.

Once you are done, run the program, you should see the following result if you enter 234561. If not, debug your code (*Shift+Ctrl+J* in Firefox).



Before going to next exercise, make sue your JS code is a good match to your flowchart for this problem. If not, you should fix this mismatch.

Exercise 12. Copy **ct_Ex11.html** and **ct_Ex11.js** to new files named **ct_Ex12.html** and **ct_Ex12.js**.

In this exercise, you translate your flowchart of Ex 12 of Part 1 to its equivalent JavaScript code.

Several changes that you need to make here are minor changes like what you did in Exercise 11 in **ct_Ex11.html** and **ct_Ex11.js**. So, we do not re-state them again here, to encourage you to make such changes independently and with minimum guidance.

In the loop of your flowchart for Ex 12 of Part 1, you may separate the digits and make the reverse of the input number along the way—i.e., in loop iterations. For instance, assume the input number is 235, before entering the loop, you should declare a variable `reverse` and initialized it to 0 as well as storing the input number in a variable, let's call it `temp`.

In the first iteration of the loop, you will have:

> 235 modulus 10 is 5; and, the reverse number that you currently have, which is 0, times 10 plus 5 is 5 and you store it in variable `reverse`; then, you reduce 235 to 23 by dividing it by 10.

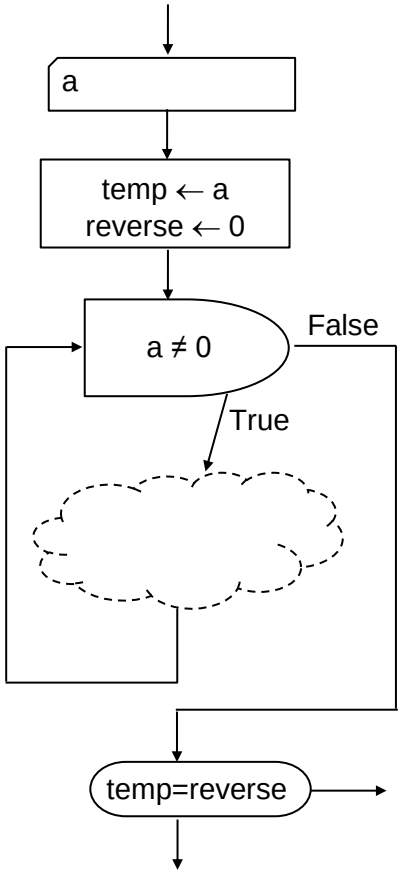
Now, in the second iteration of the loop, you will have:

> 23 modulus 10 is 3; and, the reverse number that you currently have, which is 5, times 10 plus 3 is 53 and you store it in variable `reverse`; then, you reduce 23 to 2 by dividing it by 10.

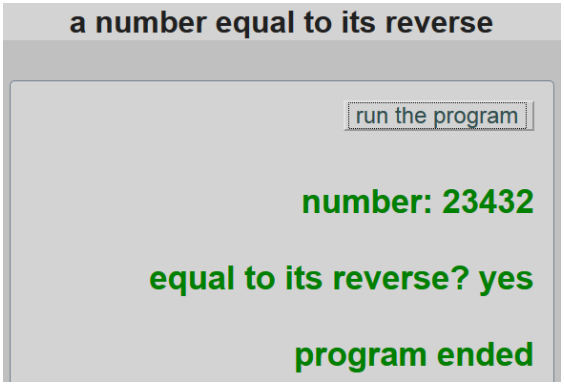
Now, in the third iteration of the loop, you will have:

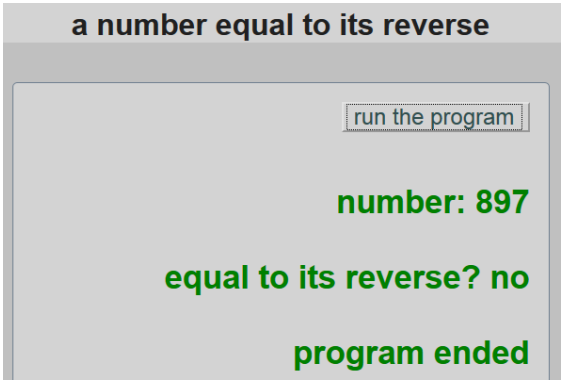
> 2 modulus 10 is 2; and, the reverse number that you currently have, which is 53, times 10 plus 2 is 532 and you store it in variable `reverse`; then, you reduce 2 to 0 by dividing it by 10.

Because the number is already reduced to zero, there is no more iteration. In other words, we exit the loop. Now, we should compare the initial number (235) by its reverse (532)—that we built within loop iterations. If they are equal, we output "yes"; otherwise we output "no". As a hint, below, we show you part of a flowchart that you may have for this exercise. You should be able to write the body of the loop based on example that we explained above.



Once you are done, run the program, you should see the following result if you enter 23432 or 897. If not, debug your code (*Shift+Ctrl+J* in JavaScript).





Before going to next exercise, make sue your JS code is a good match to your flowchart for this problem. If not, you should fix this mismatch.

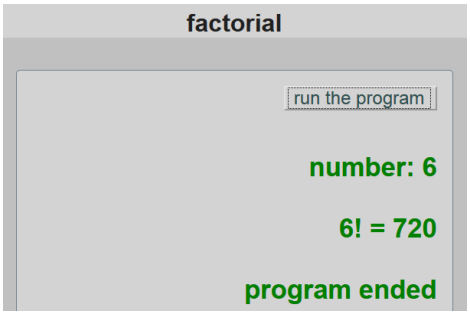
Exercise 13. Copy **ct_Ex12.html** and **ct_Ex12.js** to new files named **ct_Ex13.html** and **ct_Ex13.js**.

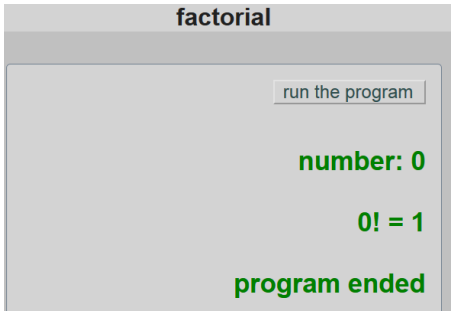
In this exercise, you translate your flowchart of Ex 13 of Part 1 to its equivalent JavaScript code.

Several changes that you need to make here are minor changes like what you did in previous exercises in **html** and **js** files. So, we do not re-state them again here, to encourage you to make such changes independently and with minimum guidance. Make sure name of the event function is `factorial()`.

The major difference between the flowchart of this exercise compare to those of previous exercises is that the number of iterations in this exercise is *deterministic*. In other words, we know how many iterations needs to be made—in advance. For instance, for calculating 5!, five iterations is required. No more no less, whereas in previous exercises we did not know how many digits the input number will have. Therefore, we did not know how many iterations the loop will need. Hence, none of the previous exercises should have been solved with a `for` loop. Ex 13, though, can be devised with a `for` loop.

Once you are done, run the program, you should see the following result if you enter 6 or 0. If not, debug your code (*Shift+Ctrl+J* in Firefox).





Before going to next exercise, make sue your JS code is a good match to your flowchart for this problem. If not, you should fix this mismatch.

Exercise 14. Copy **ct_Ex13.html** and **ct_Ex13.js** to new files named **ct_Ex14.html** and **ct_Ex14.js**.

In this exercise, you translate your flowchart of Ex 14 of Part 1 to its equivalent JavaScript code.

Several changes that you need to make here are minor changes like what you did in previous exercises in **html** and **js** files. So, we do not re-state them again here. Just make sure name of the event function is `fibonacci()`, both in your `html` and `js` files.

This exercise can be done with a `for` loop too, because—as an example—if the input is 10, the loop should iterate exactly 8 times. Note that for the first two values, we do not need to iterate; we already know they are 0 and 1.

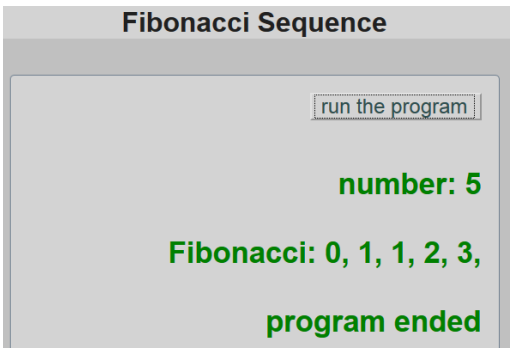
Hint: As a new value, in Fibonacci sequence, can be constructed by sum of the last and 2nd last values, you may want to declare three variables:

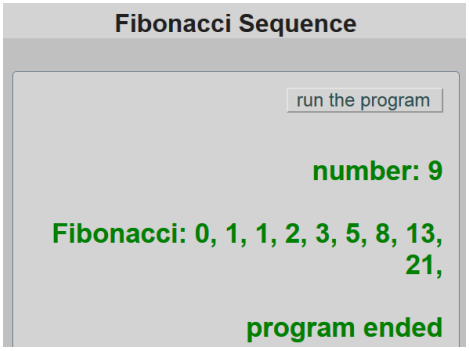
```
var secondLast=0;
var last=1;
var newValue;
```

Note that `secondLast` and `last` variables are initially 0 and 1, respectively, because that's they way Fibonacci sequence starts.

In the body of the `for` loop, we calculate `newValue` and update the values for `last` and `secondLast` variables.

Once you are done, run the program, you should see the following result if you enter 5 or 9. If not, debug your code (*Shift+Ctrl+J* in Firefox).





Before going to next exercise, make sue your JS code is a good match to your flowchart for this problem. If not, you should fix this mismatch.

Exercise 15. Copy **ct_Ex14.html** and **ct_Ex14.js** to new files named **ct_Ex15.html** and **ct_Ex15.js**.

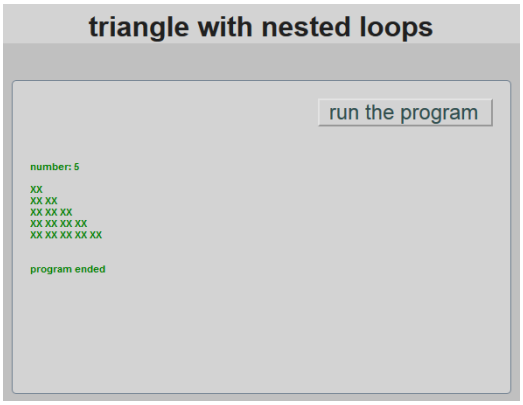
In this exercise, you translate your flowchart of Ex 15 of Part 1 to its equivalent JavaScript code.

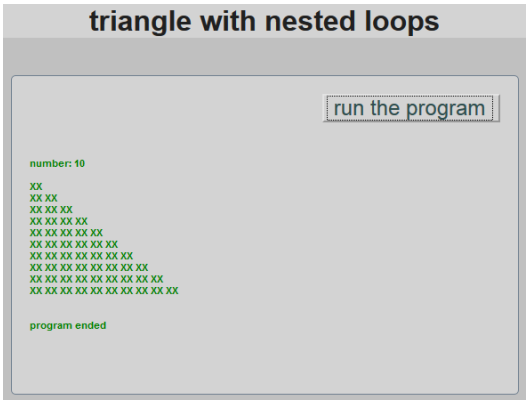
Several changes that you need to make here are minor changes like what you did in previous exercises in **html** and **js** files. So, we do not re-state them here.

This exercise can be done with `for` loop, as iterations are deterministic. For instance, if input is 5, we need to iterate exactly 5 times to output those 5 rows; also, in each row *i*, we should put *i* pairs of XX. So, we need to nest one loop inside another.

Note: in this exercise you would need to change text alignment to left. You can do this just in your `js` code. Also, you would need to make the output font size smaller, let's say to 11px. For these, you can modify the `style` property of `outputObj` just before you enter the loop, like in line 18 and 19.

Once you are done, run the program, you should see the following result if you enter 5 or 10. If not, debug your code (*Shift+Ctrl+J* in Firefox).





G. AFTER-LAB TASKS (THIS PART WILL NOT BE GRADED)

In order to review what you have learned in this lab as well as expanding your skills further, we recommend the following questions and extra practice:

1) You should revisit the 7 exercises after the lab and learn about the parts of your flowcharts that were not a good match to your `JavaScript` code.
    a. You may want to do some sort of reverse engineering here: study the `JavaScript` code that you eventually developed for each exercise and draw a flowchart for that. Be careful not to include any `JavaScript`-specific notation in your flowcharts. Flowcharts should be as independent as possible from programming languages. That means your flowchart should be understandable to anyone who knows programming even if he/she does not know any `JavaScript`.
    b. Hence, your flowchart should not use functions like `parseFloat`, `getElementById`, etc., or keywords like `if`, `else`, `var`, etc. You are also highly discouraged to use "=" for an assignment; instead you should use "←".
2) Once you have your own 7 flowcharts and corresponding `JavaScript` code polished, take photos (or screenshots) of each and add them to your `myLearningKit` webpage such that when the corresponding buttons are clicked, your solutions are shown.
3) We will provide you with sample solutions on February 7. The sample solution should NOT be used as a means of learning how to tackle those 7 exercises. Instead, it should be only used as a reference to compare your own solution with our solution and learn from differences. In general, you cannot learn much computational thinking skills, if any, by studying a solution without putting your efforts first to come up with a solution (even a non-perfect one).

Please feel free to discuss any of these questions in the course forum or see the TAs and/or Instructors for help.